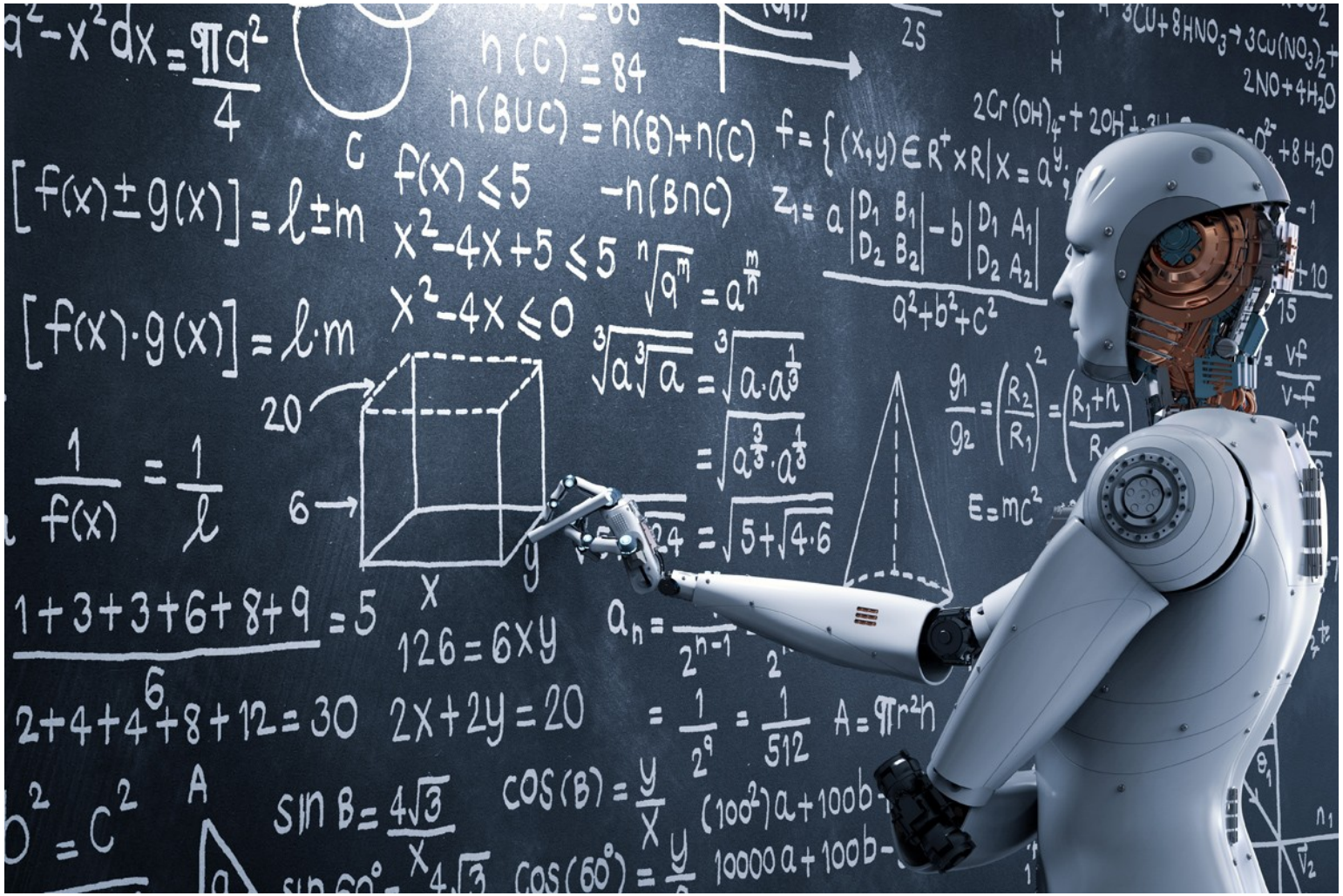
A1. Estrategias de resolución de problemas

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
886 12 04 64
ies.teis@edu.xunta.es

GOBIERNO DE ESPAÑA · MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Unión Europea-NextGenerationEU



A1. Estrategias de resolución de problemas

Índice.

1. Problema de resolver problemas 3

   1.1. ¿Qué es un problema? 4
   1.2. Resolución de problemas 5
   1.3. Definición formal de un problema 7
   1.4. Ejemplo de problema 8
   1.5. Tipos de problemas 10
   1.6. Soporte computacional 11

2. Estrategias de búsqueda de soluciones 14

   2.1. Tipos de estrategias de búsqueda 17

      2.1.1. Búsqueda en anchura 18
      2.1.2. Búsqueda en profundidad 19
      2.1.3. Búsqueda de coste uniforme 20
      2.1.4. Búsqueda en profundidad limitada 21
      2.1.5. Búsqueda en profundidad progresiva o iterada 22
      2.1.6. Búsqueda bidireccional 23
      2.1.7. Búsqueda de primero el mejor 24
      2.1.8. Método del gradiente o de ascensión de colinas 25
      2.1.9. Método del templo simulado 26

3. Problema de la satisfacción de restricciones 27

4. Juegos 31

   4.1. Algoritmo minimax 33

A1. Estrategias de resolución de problemas

1. Problema de resolver problemas.

A1. Estrategias de resolución de problemas

1.1. ¿Qué es un problema?

Un problema es:

- Una situación en la que un individuo desea hacer algo, pero desconoce el curso de la acción necesaria para lograr lo que quiere (Newel y Simon, 1972).
- Una situación en la que un individuo actúa con el propósito de alcanzar una meta utilizando alguna estrategia en particular (Chi y Glaser, 1983).
- La diferencia entre la meta (o solución) y el estado inicial (Mayer, 1983).

Según la visión de Mayer, los problemas tienen cuatro componentes:

- Metas → lo que se desea lograr en una situación determinada.
- Datos → información (numérica o verbal) con la que se cuenta al analizar un problema.
- Restricciones → factores que limitan la vía para alcanzar la solución.
- Métodos u operaciones → procedimientos utilizados para resolver el problema.

1.2. Resolución de problemas.

La resolución de un problema es:

- Un proceso cognoscitivo complejo que involucra conocimiento almacenado en la memoria a corto y a largo plazo (Dijkstra, 1972).
- Un proceso que puede describirse a partir de los siguientes elementos (Mayer, 1983):
  - Una situación en la que se quiere hacer algo, pero se desconocen los pasos precisos a seguir.
  - Un conjunto de elementos que representan el conocimiento relacionado con el problema.
  - Un solucionador de problemas (o sujeto) analiza el problema, sus metas y sus datos, se forma una representación del problema en su sistema de memoria, y opera sobre la representación del problema para reducir la discrepancia entre datos y meta.
  - La solución al problema está constituida por la secuencia de operaciones que pueden transformar los datos en metas.
  - Al operar sobre los datos y las metas, el solucionador de problemas utiliza los siguientes tipos de información:
    - Información almacenada en su memoria a largo plazo en forma de esquemas (o producciones).
    - Procedimientos heurísticos.
    - Algoritmos.
    - Relaciones con otras representaciones.
  - La búsqueda es el proceso de operar sobre una representación inicial bajo el fin de encontrar una solución al problema. La representación puede transformarse en otras representaciones.
  - La búsqueda continúa hasta alcanzar una solución o si el solucionador de problemas se da por vencido.

1.2. Resolución de problemas.

La resolución de problemas es, realmente, una búsqueda en un espacio de estados, en el que:

Estado = {Q, R, C}

Q: estructura de datos que describen el estado

R: reglas u operaciones que describen las transiciones en el espacio de estados.

C: estrategia de control

La solución consiste en encontrar una secuencia de reglas $r_1,\ldots,r_n$ que conduzcan desde el estado inicial q_0 al estado final q_f.

El estado inicial será la entrada del laberinto y el estado final la salida del mismo.

Una posible solución sería: abajo, izquierda, abajo, izquierda, arriba, … Todos estos movimientos son la secuencia para salir del laberinto.

1.3. Definición formal de un problema.

Los pasos a seguir para obtener una definición formal de un problema son los siguientes:

- Definir un espacio (o conjunto) de estados.
- Especificar uno o más estados iniciales.
- Especificar uno o más estados finales (meta/objetivo).
- Definir reglas sobre las acciones disponibles.

El problema se resuelve utilizando las reglas en combinación con una estrategia de control.

La **estrategia de control** establece el orden de aplicación de las reglas y resuelve los conflictos.

El agente resolvedor de problemas analiza las acciones que conllevan a la solución del problema.

Un problema puede definirse de forma más clara en función de:

- Estado inicial → situación actual del agente.
- Función sucesor → siguiente acción a realizar dentro de la representación abstracta de acciones posibles (acciones posibles).
- Test objetivo → determina si el estado es el adecuado o no.
- Costo del camino → costo del camino a recorrer hasta llegar al destino.

1.4. Ejemplo de problema.

Si tenemos tres jarras con capacidades de agua de 8, 5 y 3 litros, ¿qué debemos hacer si deseamos servir 4 litros de agua?

1.4. Ejemplo de problema.

Una solución sería la siguiente:

8	0	0
3	5	0
3	2	3
6	2	0
6	0	2
1	5	2
1	4	3
4	4	0

1.5. Tipos de problemas.

El marco para la definición de problemas que se acaba de plantear da cabida a un sinnúmero de problemas de naturalezas muy diversas y diferentes:

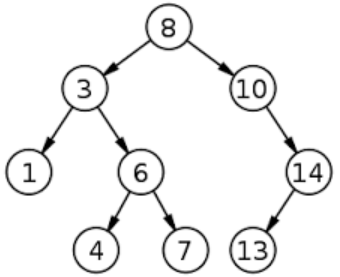
- **Problemas de un estado inicial (single-state)** → ej. cómo ir de Vigo a Pontevedra. Esto hace que el sistema busque un camino entre una ciudad inicial (no cambiante) y otra ciudad.
- **Problemas de múltiples estados iniciales (multiple-state)** → ej. dispositivos GPS a los que se les pregunta cómo ir desde un sitio a otro sitio (Vigo a Pontevedra, Sevilla a Gijón). Esto hace que el sistema busque un camino entre dos ciudades cualesquiera.
- **Problemas de contingencia (contingency)** → ej. cómo ir de Vigo a Pontevedra sin pagar peaje. Esto obliga al sistema a tener en cuenta una solución alternativa que cumpla con la restricción.
- **Problemas de exploración (online)** → ej. cómo llegar a Pontevedra desde Vigo sin saber dónde está Pontevedra. Esto obliga a recorrer (explorar) las carreteras hasta encontrar Pontevedra.

1.6. Soporte computacional.

Un **grafo** es una estructura de información compuesta de nodos (piezas de información) y de arcos (uniones entre ellos), dotados de los siguientes elementos:

- **Hojas** → nodos sin descendientes (los últimos).
- **Camino** → sucesión de nodos siguiendo los arcos.
- **Ciclo** → camino cerrado (bucle).
- **Grafo dirigido** → los arcos indican el sentido de la relación.
- **Grafo acíclico** → no contiene ciclos.
- **Grafo conexo** → si hay un camino entre dos nodos

Un **árbol** es un grafo dirigido acíclico conexo en el que:



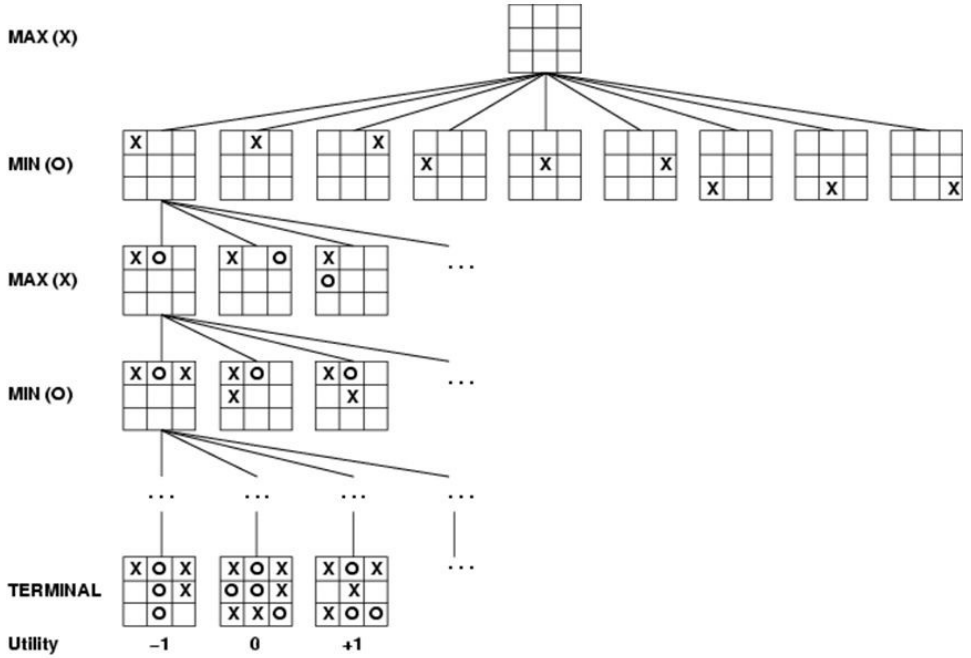
- Hay un único nodo raíz.
- Cada nodo tiene un único padre.
- Para cada nodo existe un único camino que lo conecta con el nodo raíz.
- Coste de un nodo → coste de llegar al nodo desde la raíz a lo largo del mejor camino.

1.6. Soporte computacional.

Los problemas de búsqueda se modelan a través de árboles porque permiten trabajar con el espacio de estados del problema:

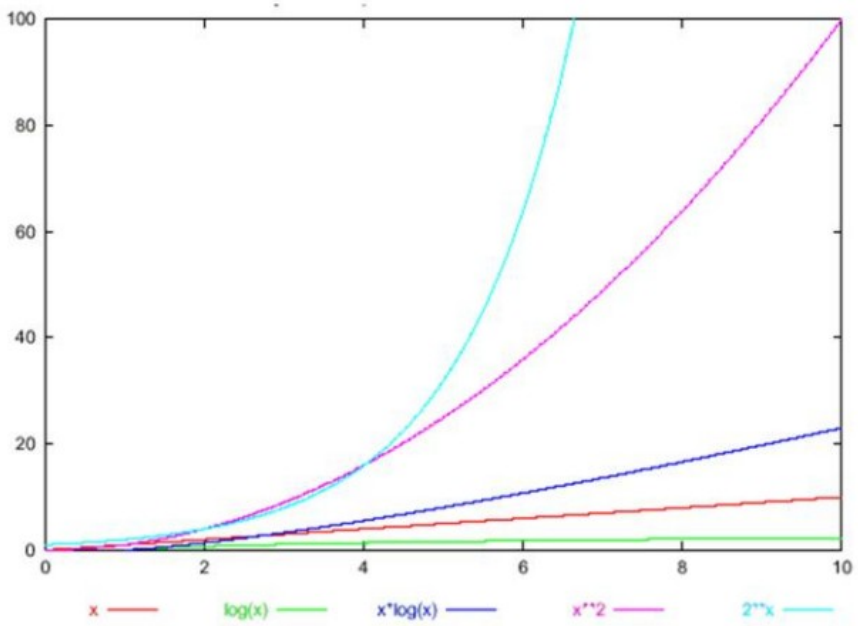
- En la raíz del árbol se hallará el estado inicial.
- En las hojas del árbol se hallarán los diferentes estados finales (incluidos los solución).
- En las ramas se hallarán los estados intermedios (nodos) por los que el problema puede pasar hasta alcanzar el estado final.
- Las transiciones entre estados (nodos) se producen al aplicar las reglas de cambio de estado definidas en el problema y expresadas e forma de operadores.

Un árbol (parcial) de búsqueda para el juego de tres en raya.



1.6. Soporte computacional.

La explosión combinatoria de los espacios de estados, a poco que los problemas a resolver tengan cierto tamaño, es uno de los grandes obstáculos de la Inteligencia Artificial → sobre todo si se aborda mediante técnicas de fuerza bruta, es decir, peinando todo el espacio de estados a ciegas hasta encontrar un estado solución.



Ante este tipo de problemas, se hace necesario el uso de otras estrategias con el fin de aminorar los tiempos de búsqueda y aplicando algún tipo de información adicional que ayude en la selección de unos caminos sobre otros en el árbol.

2. Estrategias de búsqueda de soluciones.

2. Estrategias de búsqueda de soluciones.

No sólo se ha de plantear los problemas de una manera formal, sino que es necesario encontrar la forma apropiada de decidir qué regla aplicar desde el estado inicial para llegar al estado final y el orden en el que se deben aplicar, todo esto constituyen las **estrategias de búsqueda**.

Q,V: lista de nodos
S: nodo inicial
1) Inicializar Q = {S},V = {S}, $coste_S$=0
2) Si Q está vacía, devolver SIN_SOLUCIÓN
3) Sacar un nodo N de Q
4) Si N es nodo final, devolver N como solución
5) Para todos los descendientes N_i de N:
- Enlazar N_i con N
- Si N_i no está en V:
  $coste_{Ni}$=$coste_N$+$coste_{N\rightarrow Ni}$
  añadir N_i a Q
  añadir N_i a V
- En otro caso:
  Si $coste_{Ni}$>$coste_N$+$coste_{N\rightarrow Ni}$
    $coste_{Ni}$=$coste_N$+$coste_{N\rightarrow Ni}$
    añadir N_i a Q
  En otro caso, no hay que hacer nada
6) Volver a 2

El algoritmo general de búsqueda parte de un nodo inicial, expande el nodo i-ésimo y ejecuta todas sus transiciones:

- Si el nodo destino no existe en el árbol → se añade
- Si el nodo destino ya existe:
  - Expandir el nodo i-ésimo apuntando al nodo existente.
  - Si se está registrando el mejor camino → comprobar si el camino es mejor.
- Si el nodo destino es un nodo final → devolver como solución el camino seguido desde el nodo inicial.

2. Estrategias de búsqueda de soluciones.

La diferencia entre las estrategias de búsqueda radica en el orden en el que se recorren los nodos ya descubiertos para expandirlos, así una estrategia puede decidir expandir antes los más recientemente descubiertos, mientras que otra puede preferir expandir los nodos en el mismo orden en que se descubren.

Este orden en la expansión de los nodos genera estrategias muy diferentes que se evaluarán según:

- **Completitud de la solución** → capaces de encontrar la solución.
- **Complejidad temporal** → número de nodos recorridos para encontrar la solución.
- **Complejidad espacial** → nodos en cola descubiertos esperando ser procesados.
- **Optimalidad de la solución** → solución de menor coste.

Los parámetros a evaluar en cada solución encontrada tienen que ver con la propia naturaleza del problema y NO con la estrategia elegida, y son los siguientes:

- Factor de ramificación (b).
- Profundidad de la solución (d).
- Máxima profundidad (m).

No obstante, en función de estos parámetros (es decir, de la naturaleza del propio problema) dependerá lo buena o mala que sea una estrategia para un problema dado.

2.1. Tipos de estrategias de búsqueda.

Las estrategias de búsqueda pueden ser de los siguientes tipos:

- **Sin información del dominio o búsqueda a ciegas** → sólo emplean la información en la definición del problema.

	Completo	Óptimo	Tiempo	Espacio
Búsqueda en anchura	Sí	Sí	$O(b^d)$	$O(b^d)$
Búsqueda en profundidad	No	No	$O(b^m)$	$O(b\,m)$
Búsqueda de coste uniforme	Sí	Sí	$O(b^{1+(C^*/e)})$	$O(b^{1+(C^*/e)})$
Búsqueda en profundidad limitada	No	No	$O(b^p)$	$O(b\,p)$
Búsqueda en profundidad progresiva	Sí	Sí	$O(b^d)$	$O(b\,d)$
Búsqueda bidireccional	Sí	No	$O(b^{d/2})$	$O(b^{d/2})$

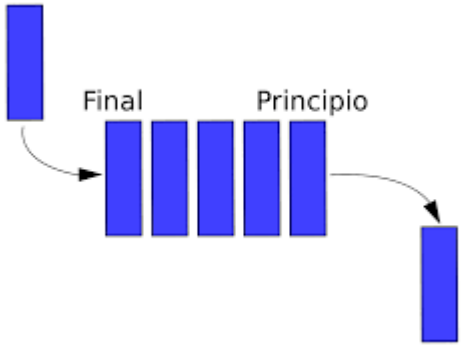
- **Con información del dominio o estrategias heurísticas** → emplean información del espacio de búsqueda para evaluar (heurística) el coste de acceso a cada nodo.
  - Primero el mejor (best-first).
    - Búsqueda avariciosa (greedy search).
    - Búsqueda A*.
  - Método del gradiente o de ascensión de colinas (hill-climbing).
  - Método del templo simulado (Simulated annealing).

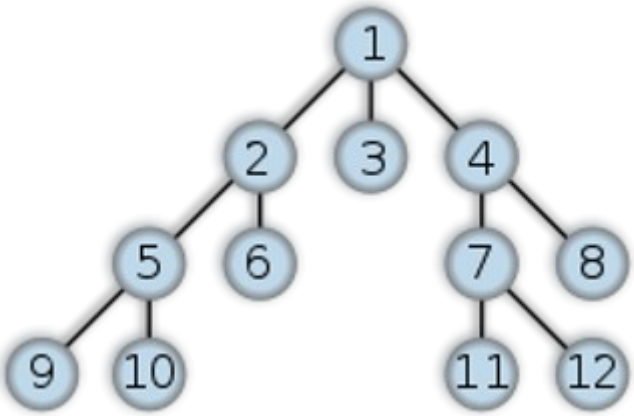
2.1.1. Búsqueda en anchura.

La búsqueda en anchura consiste en buscar horizontalmente los nodos, es decir, expandir todos los nodos de un mismo nivel, para así poder pasar al siguiente nivel.

Es una búsqueda óptima si el espacio de estados no es infinito, pero tiene una complejidad de espacio debido a que guarda en memoria los nodos extendidos.

Presenta una estructura de cola FIFO → extrae por el principio y añade por el final.



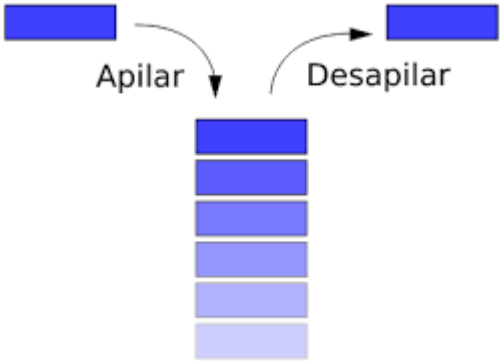


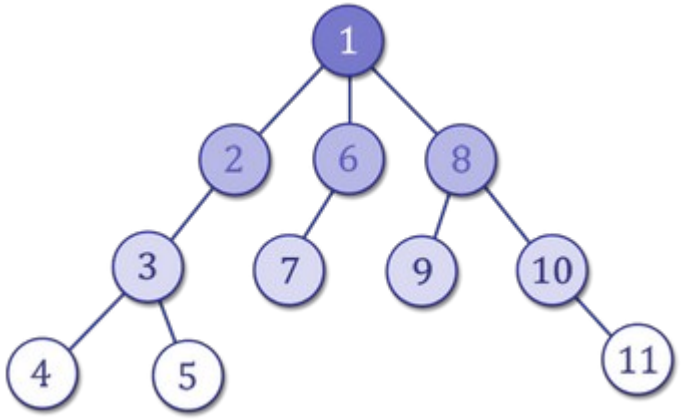
2.1.2. Búsqueda en profundidad.

La búsqueda en profundidad consiste en buscar verticalmente, es decir, el nodo raíz se expande, y después su hijo de forma vertical y por la izquierda.

Si el nodo está repetido, se pasa al siguiente nodo de la derecha y esto se hace hasta terminar al último nodo de la rama.

Presenta una estructura de cola LIFO → extrae y añade por el final.

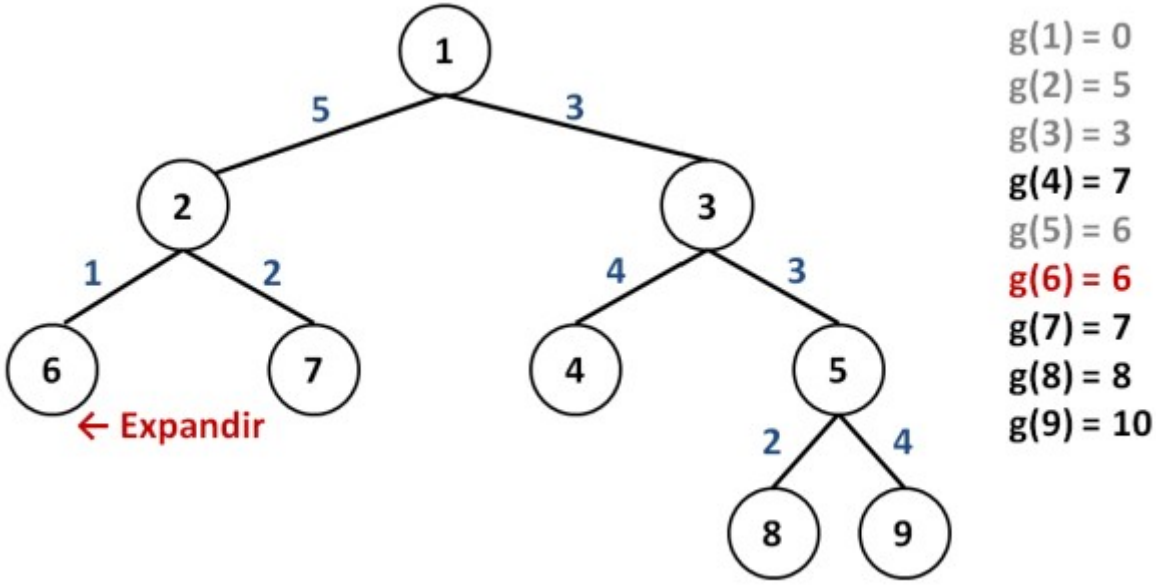




2.1.3. Búsqueda de coste uniforme.

La búsqueda de coste uniforme consiste en expandir los nodos que tengan un coste de camino menor, por consiguiente, esta búsqueda no se enfoca en el número de pasos a seguir sino más bien, en el peso asociado a los pasos.

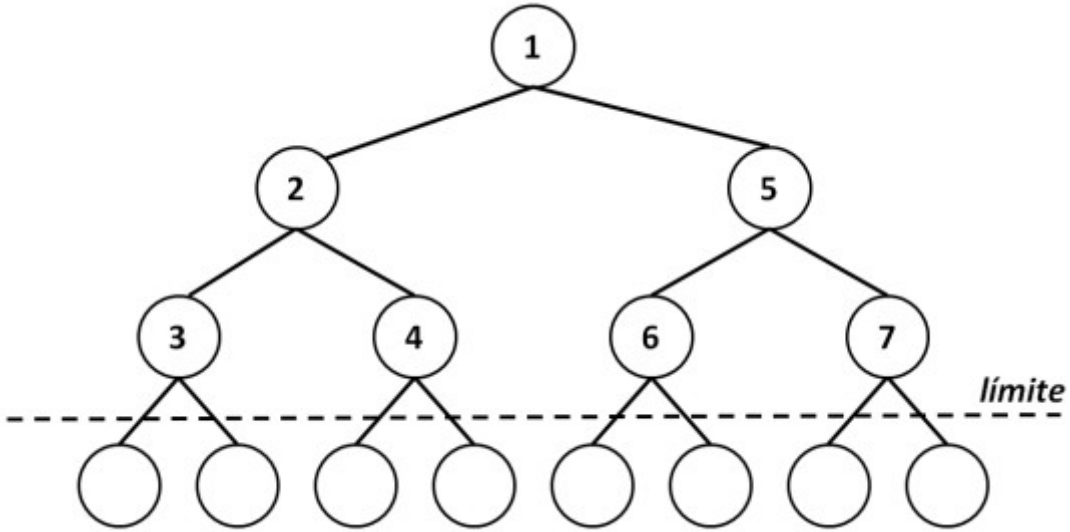
Esta búsqueda trabaja con grafos binarios y su funcionamiento consiste en asignar un coste al camino que recorre, pero se relaciona mucho con una búsqueda en anchura aunque se diferencia en la asignación de un coste para la elección de las acciones a desarrollar.



2.1.4. Búsqueda en profundidad limitada.

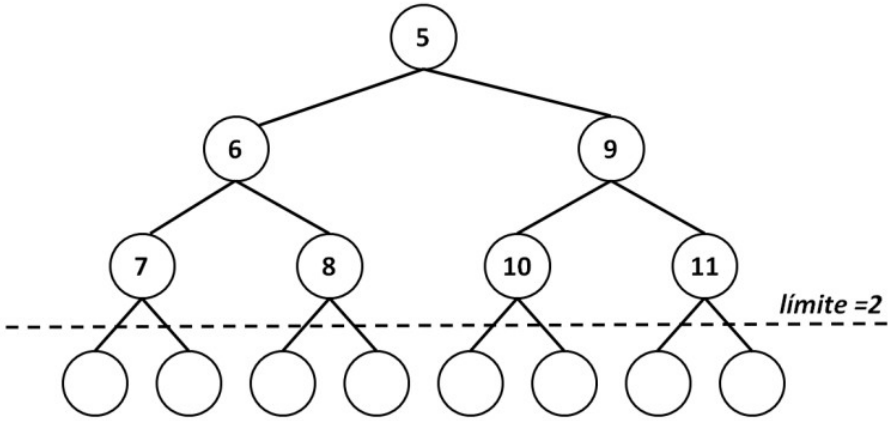
La búsqueda en profundidad limitada consiste en aplicar la búsqueda en profundidad con un límite en profundidad p predeterminado, impidiendo caminos infinitos.

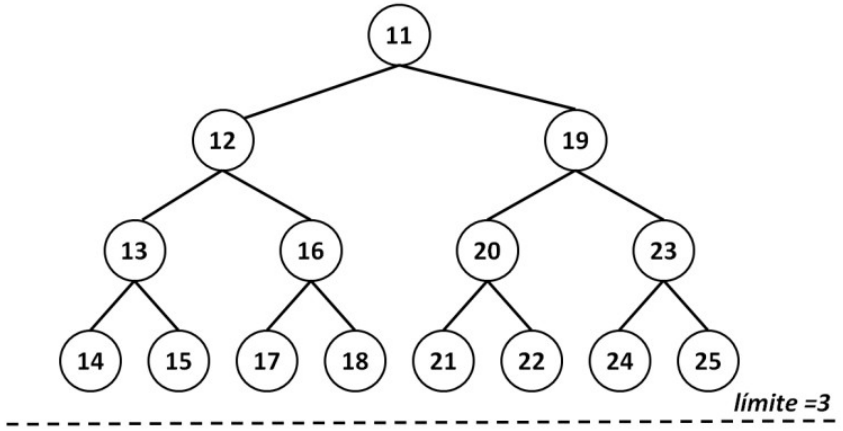
Los nodos a profundidad p se tratan como si no tuviesen ningún sucesor.



2.1.5. Búsqueda en profundidad progresiva o iterada.

La búsqueda en profundidad progresiva o iterada consiste en repetir la búsqueda en profundidad limitada, incrementando gradualmente el límite hasta encontrar el objetivo.

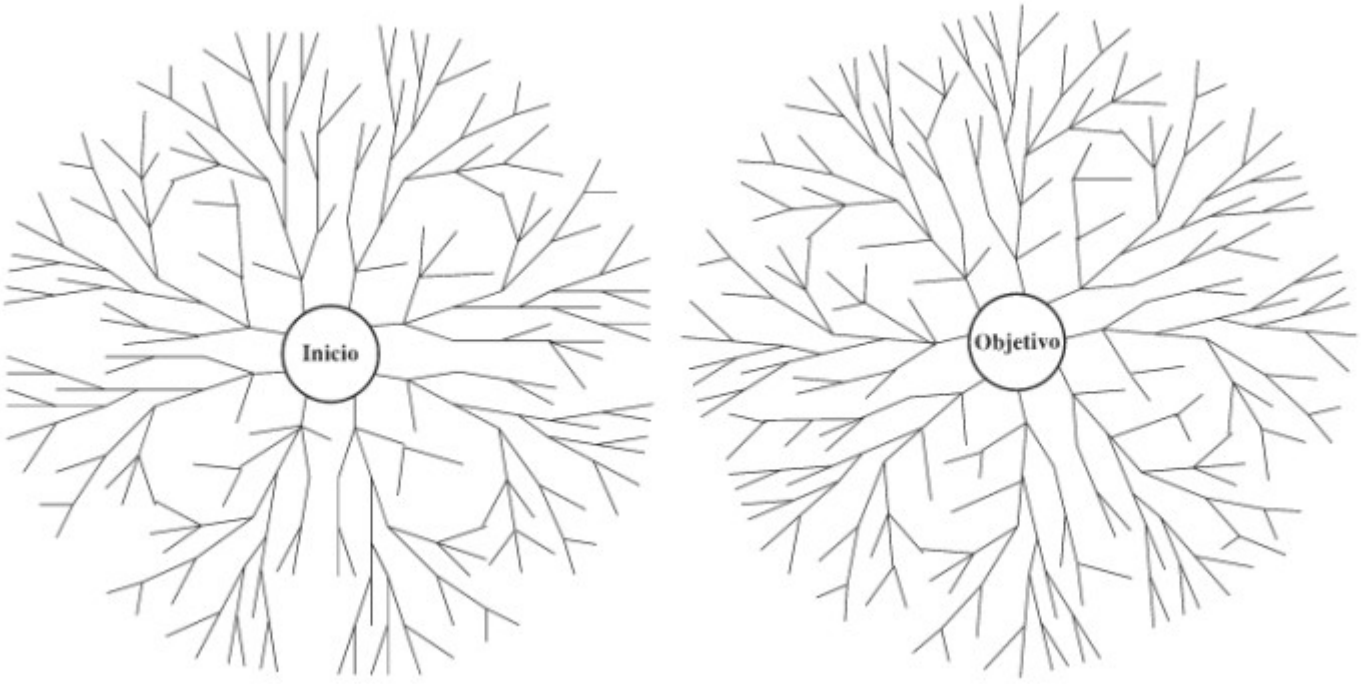




2.1.6. Búsqueda bidireccional.

La búsqueda bidireccional consiste en la ejecución de dos búsquedas de forma simultánea (hacia adelante desde el estado inicial y hacia atrás desde el objetivo) hasta que las dos búsquedas se encuentren.

Antes de expandir un nodo, se verifica si está en el otro árbol de búsqueda.



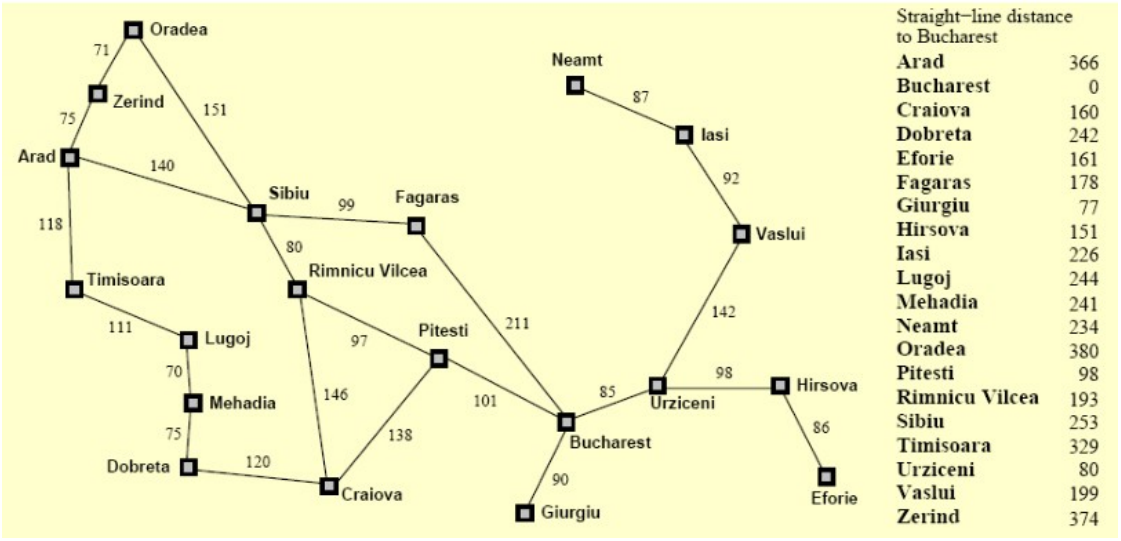
2.1.7. Búsqueda de primero el mejor.

La búsqueda del primero el mejor utiliza una función de evaluación (heurística) para cada nodo y se expande el nodo mejor evaluado no expandido (misma idea que en la búsqueda del coste uniforme).

Dos variantes de esta estrategia son:

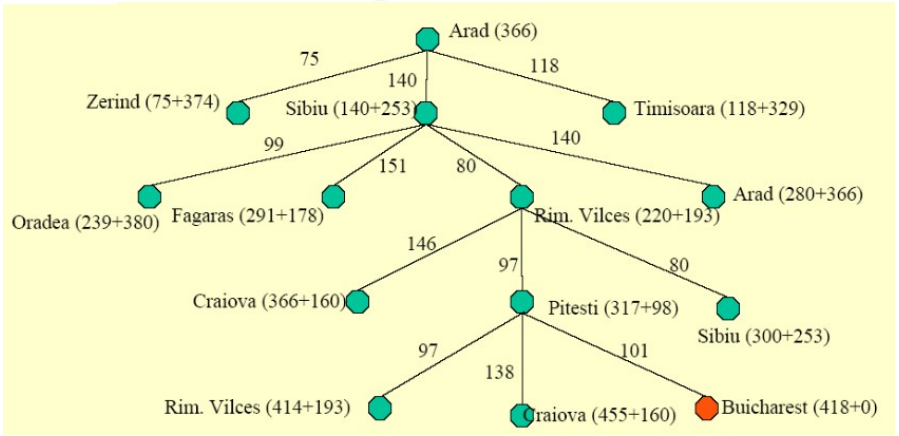
- **Búsqueda avariciosa (Greedy Search)** → la función de evaluación estima el coste del nodo-i hasta la meta, con lo que se expande el nodo que parece estar más cerca de la meta. Una buena función de evaluación puede mejorar drásticamente la búsqueda.
- **Búsqueda A*** → evita expandir nodos que resultan muy costosos. Alcanza la solución óptima siempre que se utilice una heurística admisible que no sobreestime el coste real.

Búsqueda de Greedy

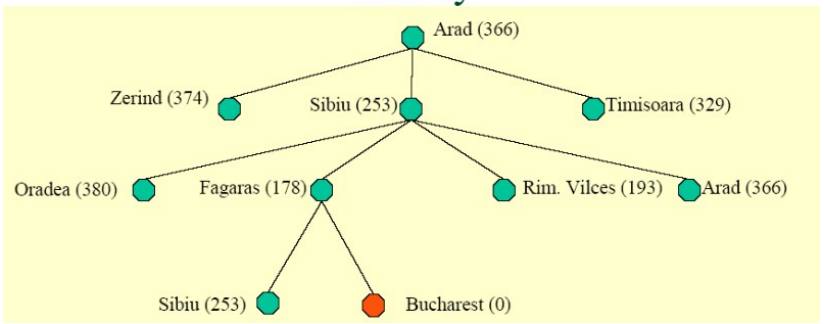


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Búsqueda A*



Greedy



2.1.8. Método del gradiente o de ascensión de colinas.

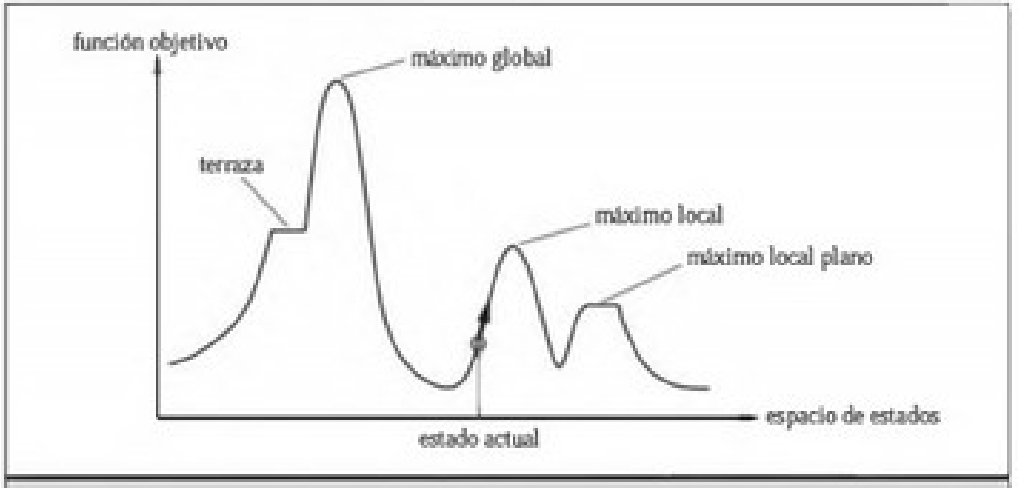
La búsqueda por el método de ascensión de colinas (hill-climbing) consiste en moverse hacia un punto mejor valorado que el actual, sino se evalúa otro punto vecino.

La función heurística determina la calidad del resultado y la rapidez de la búsqueda.

El método acaba cuando llega a la meta o si no hay más mejoras.

Los problemas que puede tener son:

- Máximo local → si todos los vecinos tienen una función heurística peor.
- Meseta → si todos los vecinos tienen la misma función heurística que el nodo actual.
- Crestas → las crestas causan una secuencia de máximos locales que dificultan la navegación para los algoritmos ávaros.



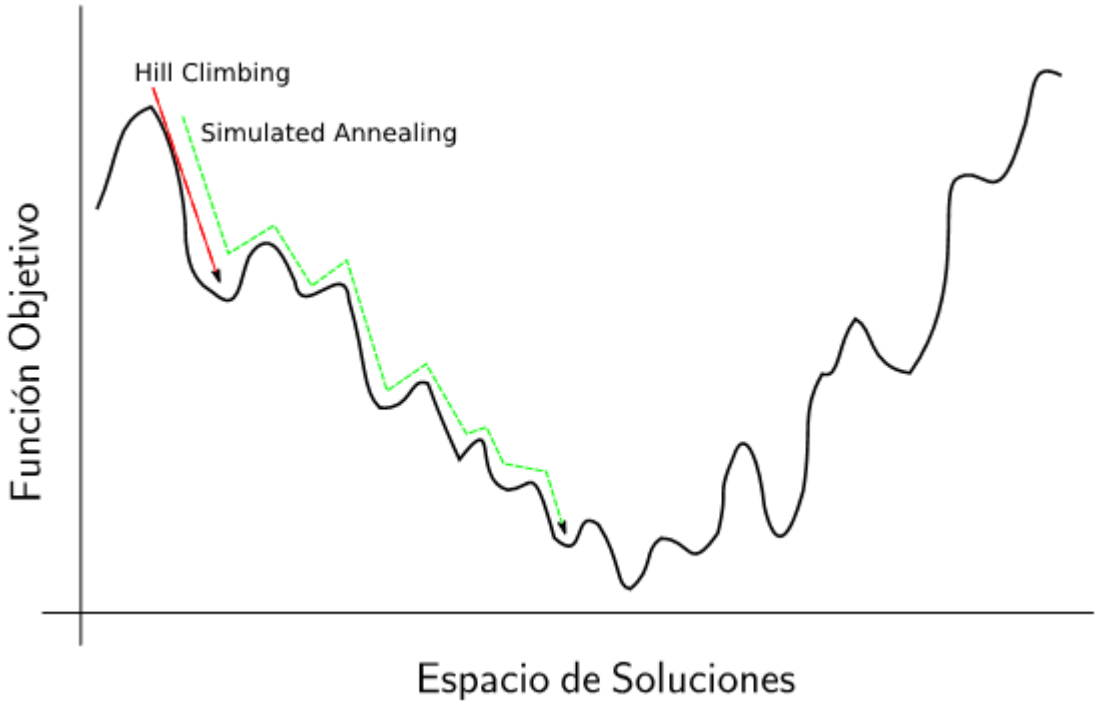
2.1.9. Método del templo simulado.

La búsqueda por el método del templo simulado se basa en la idea de escapar de los máximos locales, permitiendo movimientos “incorrectos”, pero reduciendo gradualmente su tamaño y frecuencia.

Este método es bastante parecido al de la ascensión de colinas, pero el movimiento escogido se realiza de forma aleatoria, y si mejora la situación es aceptado.

Este método utiliza como parámetro la temperatura T del proceso. Si dicha temperatura se reduce muy lentamente, se alcanza la solución óptima.

Se desarrolló en 1953 para el modelado de procesos físicos.



3. Problema de la satisfacción de restricciones.

3. Problema de la satisfacción de restricciones.

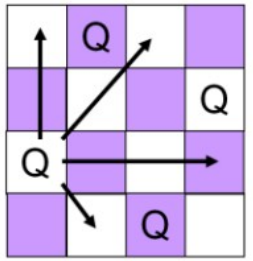
El **objetivo** consiste en descubrir un estado del problema que satisfaga un conjunto de restricciones.

Los CSP (Constraint Satisfaction Problem) son una serie de problemas especiales en los que:

- Los estados están definidos por una serie de valores asignados a un conjunto de variables.
- El objetivo está definido por una serie de restricciones en los valores de las variables.

Ejemplo clásico: 4 reinas

- Poner cuatro reinas (damas) sin que ninguna pueda atacar a las demás
  (→ sin que haya dos en la misma columna y no estén en diagonal)



- Variables: Q_1, Q_2, Q_3, Q_4
- Dominio: {1,2,3,4}
- Restricciones:
  - Q_i != Q_j
  - $|Q_i - Q_j|$!= $|i - j|$
- Significado: (Q_1, Q_2) pueden valer (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)

3. Problema de la satisfacción de restricciones.

Los problemas de satisfacción de restricciones son de lo más común que hay en la vida diaria y, en general, suelen ser problemas que involucren un reparto de recursos, siendo limitados los grados de asignación de los mismos.

Podemos imaginarnos un espacio de estados inicialmente enorme, al que se van recortando pedazos con cada nueva restricción incorporada.

Algunos ejemplos son los siguientes:

- Planificación de horarios de clase.
- Configuración de hardware incompatible.
- Planificación de rutas.
- Planificación de producción.
- Problemas de asignación de recursos.
- Evaluación de riesgos en inversiones.
- Asignación de invitados a un banquete por mesas.
- Elección del lugar para pasar las vacaciones.
- Reparto de tareas entre los componentes de un grupo de trabajo.

3. Problema de la satisfacción de restricciones.

En la búsqueda de la satisfacción de restricciones el enfoque inicial es el de una búsqueda normal:

- Estado inicial → variables sin asignar.
- Operadores → asignar valores a variables no asignadas.
- Objetivo → todas las variables asignadas y cumpliendo todas las restricciones.
- Estrategia → cualquiera, por ejemplo, búsqueda en profundidad.

Desventajas → baja eficiencia porque el orden de asignación es irrelevante y no se comprueban las restricciones no cumplidas.

Algunas mejoras son las siguientes:

- Búsqueda con retroceso (backtracking) → algoritmo básico sin información del dominio.
- Comprobación hacia delante (forward checking) → evita asignaciones erróneas.
- Búsqueda heurística → toma de decisiones más “inteligentes” que mejora el proceso de búsqueda.

4. Juegos.

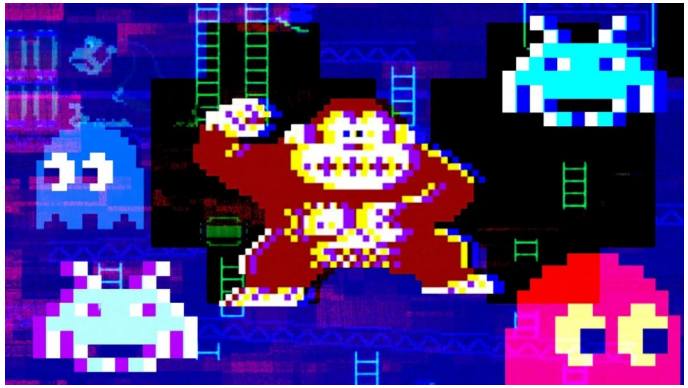
4. Juegos.

Problemas de búsqueda en los que interviene, al menos, un adversario:

- Tus movimientos, por sí mismos, no aseguran la victoria → es necesario una estrategia de oposición.
- En general, el tiempo disponible para cada movimiento impone soluciones aproximadas → sin fuerza bruta.
- Dos tipos de juegos:
  - Deterministas → no interviene el azar (4 en raya, ajedrez, damas).
  - No deterministas → el azar está presente (backgammon, parchís, monopoly, ...)

La inteligencia artificial en los juegos gusta tanto debido a que es:

- Divertida.
- Difícil.
- Fácil de formalizar y con un número pequeño de acciones.



4.1. Algoritmo minimax.

El algoritmo minimax tiene por objetivo elegir el mejor movimiento para uno mismo (MAX), suponiendo que el adversario (MIN) tratará de minimizar su pérdida.

Los pasos del algoritmo son los siguientes:

- Generar el árbol de juego hasta los nodos finales.
- Calcular la función de utilidad de cada nodo final.
- Propagar hacia arriba para generar nuevos valores de la función de utilidad para todos los nodos.
  - Minimizando para MIN.
  - Maximizando para MAX.
- Elegir como jugada a realizar aquel movimiento que conduzca al nodo final con mayor función de utilidad.

El problema al que se enfrenta minimax es que el número de estados del juego es exponencial con respecto al número de movimientos, y esto, en la mayoría de juegos hace inviable generar un árbol completo debido a la limitación de recursos de memoria.

Solución → no examinar todos los estados.